MICROPROCESSORS 605-144

INSTRUCTIONS FOR DEBUG

SUMMARY OF DEBUG FUNCTIONS

FUNCTION SYNTAX	FUNCTION NAME	
A[address] - - - - - - - - - - - - -	Assemble	pp.6-8
Crange address - - - - - - - - - - -	Compare	p.8
D[address][Lvalue]- - - - - - - - -	Dump	pp.8-10
D[range]		
Eaddress[list]- - - - - - - - - - -	Enter	pp.10-12
Frange list- - - - - - - - - - - - -	Fill	p.12
G[=address1][address2...] - - - - -	Go	pp.13-14
Haddress address - - - - - - - - - -	Hex	p.14
Ivalue - - - - - - - - - - - - - - -	Input	p.15
L[address[drive record record]] -	Load	pp.15-16
Mrange address - - - - - - - - - - -	Move	p.17
Nfilespec[filespec...]- - - - - - -	Name	pp.17-18
Ovalue byte- - - - - - - - - - - - -	Output	p.18
P[=address][value]- - - - - - - - -	Trace Special	p.23
Q- - - - - - - - - - - - - - - - - -	Quit	
R[register]- - - - - - - - - - - - -	Register	pp.19-20
Srange list- - - - - - - - - - - - -	Search	p.21
T[=address][value]- - - - - - - - -	Trace	pp.21-23
U[address][Lvalue]- - - - - - - - -	Unassemble	pp.23-24
U[range]		
W[address[drive record record]] -	Write	pp.25-26

NOTES:

Items in [ ] (brackets) are optional.
Spaces are mandatory wherever shown in syntaxes above.
Enter the character "L" or "=" wherever shown in syntaxes above.
All function commands are entered by pressing the RETURN key.
Leading zeros are not required in any DEBUG command statements.

REGISTER FLAG CODES IN DEBUG

FLAG NAME	SET CODE	CLEAR CODE
Overflow	OV Overflow - - - - -	NV No overflow
Direction	DN Decrement- - - - -	UP Increment
Interrupt	EI Enabled- - - - - -	DI Disabled
Sign	NG Negative - - - - -	PL Plus
Zero	ZR Zero - - - - - - -	NZ Not zero
Auxiliary	AC Auxiliary carry- -	NA No aux. carry
Parity	PE Even - - - - - - -	PO Odd
Carry	CY Carry- - - - - - -	NC No carry

ENTRY COMMANDS

ENTRY FORM

DEBUG is called into operation by entering a command after the system prompt in the following format:

DEBUG [filespec [arglist]] then press **ENTER** key

where: DEBUG calls up the program DEBUG and causes the DEBUG prompt (-) to appear.

filespec is an optional file specification identifying the file to be debugged.

arglist is an optional listing of file name functions and parameters that are applied to filespec.

For example, if just a filespec is given, as follows:

DEBUG MYFILE.EXT

DEBUG loads the filespec MYFILE.EXT into memory in the lowest available segment. The BX:CX registers are loaded with the number of bytes placed into memory and the DEBUG prompt, a hyphen (-), appears.

An arglist may be specified if filespec is present. These are file name parameters and switches that are to be passed to the program filespec. Thus, when filespec is loaded into memory, it is loaded as if it had been called up with the command:

filespec arglist

Here, filespec is the file to be debugged, and the arglist is the rest of the command line, consisting of functions and parameters that are used when filespec is called up and loaded into memory

If you wish to use DEBUG without an existing file, and therefore do not wish to specify a filespec, then enter your DEBUG command as follows:

DEBUG

and press **ENTER**.

DEBUG returns the - (hyphen) prompt, signaling that it is ready to accept your functions. Since no file name has been specified, current memory, disk sectors or disk files can be worked on by invoking subsequent functions.

PRELIMINARY CONCEPTS

DEBUG is designed to create or alter binary files, just as an editor or word processor program is used to alter text, or ASCII, files. DEBUG often eliminates the need to reassemble a program to see if a problem has been corrected by a minor change. It allows you to alter the contents of a file or the contents of a CPU register and then to immediately re-execute a program to check on the validity of the changes.

All DEBUG functions (summarized in a table on page 4) may be aborted at any time by holding down the CTRL key and pressing the "C" key (CTRL-C). CTRL-S suspends displays so that you can read parts of them before they scroll away off the top of the screen. Pressing any key other than CTRL-C restarts the display. All of these entries are consistent with the control character functions available at the MS-DOS command level.

When DEBUG is called up, it sets up a program header at offset 0000 in its program work area, a code segment whose base is indicated by the CS register. If no filespec has been given to DEBUG, this program header may be overwritten without causing problems. If you are debugging a .COM or .EXE file, however, you must not change the contents of this header below offset 005CH. Do not attempt to re-execute a program once the "Program terminated normally" message is given. The program must be reloaded with the N(Name) and L(Load) functions in order for it to run properly.

FUNCTIONS

Each DEBUG function consists of a single letter followed by one or more parameters. If a syntax error occurs in a DEBUG function command line, DEBUG reprints the function command line and indicates the error with a ^ (caret) and the word "Error." For example:

```
DCS:100 CS:110
    ^ Error
```

All function commands and parameters may be entered in either upper- or lower-case, or any combination of the two.

The DEBUG functions are summarized in the table on page 4 and are described in detail, with examples, in alphabetical order on later pages.

Continued on next page --

FUNCTIONS
PARAMETERS

Continued from page 3 --

SUMMARY OF DEBUG FUNCTIONS

FUNCTION SYNTAX	FUNCTION NAME	
A[address]	Assemble	pp.6-8
Crange address	Compare	p.8
D[address][Lvalue]	Dump	pp.8-10
D[range]		
Eaddress[list]	Enter	pp.10-12
Frange list	Fill	p.12
G[=address1][address2...]	Go	pp.13-14
Haddress address	Hex	p.14
Ivalue	Input	p.15
L[address[drive record record]]	Load	pp.15-16
Mrange address	Move	p.17
Nfilespec[filespec...]	Name	pp.17-18
Ovalue byte	Output	p.18
P[=address][value]	Trace Special	p.23
Q	Quit	
R[register]	Register	pp.19-20
Srange list	Search	p.21
T[=address][value]	Trace	pp.21-23
U[address][Lvalue]	Unassemble	pp.23-24
U[range]		
W[address[drive record record]]	Write	pp.25-26

NOTES:
Items in [ ] (brackets) are optional.
Spaces are mandatory wherever shown in syntaxes above.
Enter the character "L" or "=" wherever shown in syntaxes above.
All function commands are entered by pressing the ENTER key.
Leading zeros are not required in any DEBUG command statements.

FUNCTION PARAMETERS

address A two-part designation consisting of one of the following:

- an alphabetic segment register designation plus an offset value. For example:

  CS:0100 (See notes below)

- a four-digit segment address, a colon delimiter, and an offset value. For example:

  04BA:0100

Continued on next page --

PARAMETERS

Continued from page 4 --

The segment designation or segment address may be omitted, in which case the default segment for the specified function is used. DS is the default segment for all commands except A, G, L, T, U and W, for which the default segment is CS.

byte A two-digit hex value to be placed in or read from an eight-bit address or register.

drive A one-digit hex value to indicate which disk drive a file will be loaded from or written to. Values correspond to drive letter designations as follows:

0 = A Drive; 1 = B Drive; 2 = C Drive; 3 = D Drive; etc.

list A series of byte values, or a string. List must be the last parameter in the command line. For example:

```
FCS:100 42 45 52 54 41
```

range Either two offset addresses separated by a space, such as address address, for example:

CS:100 110 or,

one address, the letter "L", and a value, such as address Lvalue, where value is the number of bytes or lines the function should operate on, for example:

```
CS:100 L10
```

The following example is invalid:

```
2000:100 4000:110
         ^ Error
```

The limit for range is 10000H. To specify a value of 10000H within the four digits available, enter 0000 (or 0).

record A one- to four-digit hex value used to indicate the absolute sector number on a disk, or the number of disk sectors to be written or loaded.

register A one- or two-digit alphabetical register designation. For example:

AX or CS or F (flag register)

Continued on next page --

PARAMETERS

Continued from page 5 --

string — Any number of characters enclosed in quote marks. The characters in a string are represented as a list of ASCII byte values. Quote marks may be either single (') or double ("). Within strings, the opposite set of quote marks may be used freely. If the delimiter quote marks that enclose the string must appear within the string, they must be doubled (repeated) wherever used. For example, the following strings are valid:

```
'This "string" is okay.'
"This 'string' is okay."
"This ""string"" is okay."
'This ''string'' is okay.'
```

However, the following strings are invalid:

```
'This 'string' is not okay.'
"This "string" is not okay."
```

Note that the double quotations are not necessary in the following strings:

```
"This ''string'' is not necessary."
'This ""string"" is not necessary.'
```

value — A one- to four-digit hex value used to specify an I/O port number or the number of times a command should repeat its functions.

NOTES: Leading zeros are not required in any DEBUG command. For example:

04CA:0000 will be interpreted the same as 4CA:0

Only hexadecimal numbers are used in DEBUG. However, the hex suffix letter "H" must not be used in DEBUG commands.

The colon ":" is always required between a segment designation (whether numeric or alphabetic) and an offset address.

ASSEMBLE (A)

Syntax:

A[address]

Purpose:

To assemble 8086/8087/8088 assembly language instructions and store the object code directly into memory.

Continued on next page --

Continued from page 6 --

Additional Notes:

All numeric values are hexadecimal (without the "H" suffix) and may be entered as 1 to 4 characters. If a syntax error is encountered, DEBUG responds with

```
^ Error
```

and redisplays the current offset address where the object code will be stored in memory.

Prefixes, such as REP, REPZ and REPNE, may be entered in front of the opcode to which they refer, either on the same line, or on a separate preceding line.

Segment override prefixes, such as CS:, DS:, ES: and SS: **must** be entered on a separate line immediately preceding the line to which the override refers.

String manipulation opcodes **must** explicitly state the size of string elements. Thus, such opcodes as MOVSB, LODSW, STOSB, SCASW and CMPSB must be used in place of MOVS, LODS, STOS, SCAS and CMPS.

The assembly language opcode for the far return is **RETF**.

DEBUG cannot always tell whether an operand refers to a word memory location or a byte memory location. Therefore, in those cases the data size **must** be explicitly stated with the prefixes "**WORD PTR**" OR "BYTE PTR." DEBUG will also accept the abbreviations "WO" and "BY." For example:

```
NEG  BY PTR [137]
DEC  WO PTR [SI]
```

DEBUG distinguishes immediate operands from memory locations by enclosing memory locations in square brackets. For example:

```
MOV  AX,5A      ;Load AX with 5AH
MOV  AX,[5A]    ;Load AX with the contents of memory
                ;location 5AH
```

The DB and DW directives will assemble byte values and word values, respectively, directly into memory.

DEBUG accepts all forms of the register indirect addressing mode instructions. For example:

```
ADD   BX,34[BP+2].[SI-1]
POP   [BP+DI]
PUSH  [SI]
```

Continued on next page --

ASSEMBLE (A)
COMPARE (C)
DUMP (D)

Continued from page 7 --

DEBUG accepts all opcode synonyms. For example,

```
LOOPZ   100  is the same as:
LOOPE   100

JA      200  is the same as:
JNBE    200
```

COMPARE (C)

Syntax:

Crange address (Default is Data Segment)

Purpose:

Check for a match between the portion of memory contents specified by range and a portion of the same size beginning at address.

Additional Notes:

If the two areas of memory are identical, there is no display and DEBUG returns with the MS-DOS system prompt. If there are differences, they are displayed as:

```
address byte1 byte2 address2
```

The following examples both have the same effect of comparing the block of memory from 100H to 1FFH with the block of memory from 300H to 3FFH:

```
C100 1FF 300   or
C100 L100 300
```

DUMP (D)

Syntax:

```
D[address][ Lvalue]
D[range]
```

Purpose:

Beginning at the address specified, display the memory contents of either a single address, a range of addresses or the number of lines specified by value.

Continued on next page --

DUMP (D)

Continued from page 8 --

Additional Notes:

If only a single address is specified, the contents of 128 bytes are displayed.

If a range of addresses is specified, the contents of the range are displayed.

If the Dump function is entered without parameters, the contents of 128 bytes are displayed beginning at an address on an even 16-byte boundary (0 in the units digit of hex offset address) at or just below the current offset address in the Data Segment (whose base is in DS). Each subsequent Dump command (entered without parameters) displays the 128 bytes immediately following those last displayed, whether or not they began on an even 16-byte boundary.

Each line of the dump is divided into three parts: first is the starting **Physical Address** (segment register contents and the offset address) of a 16-byte portion of memory containing the bytes displayed on that line; this is followed by a **Hex Dump** of up to 16 hex byte values (2 hex digits each) contained in that portion of memory, in the order of ascending offset addresses; the third part of the dump line is an **ASCII Dump**, which displays the ASCII characters corresponding to the bytes in the Hex Dump. The Hex Dump is split in half by a hyphen (-) between the eighth and ninth bytes. Nonprintable ASCII values are displayed as periods (.) in the ASCII Dump.

Each line of the dump display begins with an address on an even 16-byte boundary (0 in units digit of hex offset address). If the initial address in the dump command is not on an even 16-byte boundary, the first and/or last line of the dump may contain fewer than 16 bytes.

The following examples will illustrate the Dump function:

If the following command is entered:

DCS:105 11F or DCS:105 L1B then Dump displays 27

bytes as:

```
0D8B:0100                6E 6F 72-74 68 65 61 73 74 20 57      NORTHEAST W
0D8B:0110 49 53 43 4F 4E 53 49 4E-20 00 FF 00 FF AA AA AA ISCONSIN .......
```

If you enter the command:

DDS:10A L20 or DDS:10A 129

the display is formatted as illustrated above, but the first byte will be from offset 10AH in the Data Segment and 20H (32) bytes will be displayed.

Continued on next page --

DUMP (D)
ENTER (E)

Continued from page 9 --

If you enter the command:

DD8B:400 405 or DD8B:400 L6

the display will be formatted as illustrated above, but the bytes from physical addresses 0D8B:0400 through 0D8B:0405 will be displayed. Note that these bytes could be in any memory segment (CS, DS, ES or SS), because the segment has not been specified.

ENTER (E)

Syntax:

Eaddress[list]

Purpose:

Enter an address, display its contents and, if desired, replace its contents with new values.

Additional Notes:

If the optional list of hex values is entered, the values in the list automatically replace the existing memory contents starting at the specified address. If an error occurs, no values are changed.

If the address is entered without the optional list, DEBUG displays the address and its contents, ending with a period prompt sign, then waits for your input.

While DEBUG waits for your input, you can perform any of the following actions:

1. Replace a byte value with a new value by typing the new value after the period prompt sign. If the value you enter is invalid or has more than two digits, the invalid or extra characters will not be echoed on the CRT screen, thus indicating an error.

2. Press the SPACE bar to advance to the next byte (next memory location). To change its value, simply enter the new value, as in 1. above.

Continued on next page --

Continued from page 10 --

To advance further, continue to press the **SPACE** bar as many times as desired, once for each byte. If you advance beyond an eight-byte boundary, DEBUG starts a new display line, with the new current offset address at the beginning of the line.

3. Enter a hyphen (-) to return to the preceding byte (memory location). When the hyphen is typed, a new display line is started, with the preceding offset address and its byte contents displayed. To change the value, simply enter the new value, as in 1. above. Then, if desired, proceed to a new location with the SPACE bar or hyphen key.

4. Press the **ENTER** key at any byte position to terminate the Enter function.

For example, if the following command is entered:

```
ECS:400
```

DEBUG might display the following:

```
0D8B:400 DF.
```

To change this byte value to 58H, enter "58" and see:

```
0D8B:400 DF.58
```

To step through three more bytes, at offset addresses 0401, 0402 and 0403, press the **SPACE** bar three more times to see:

```
0D8B:400 DF.58     20.     FF.     8D.
```

To change 8DH at offset 0403 to 9EH, enter "9E" and see:

```
0D8B:400 DF.58     20.     FF.     8D.9E
```

Now, to change the 20H at offset 0401 to 4AH, enter a hyphen twice to see:

```
0D8B:400 DF.58     20.     FF.     8D.9E-
0D8B:402 FF.-
0D8B:401 20.
```

then replace the 20H by entering "4A" to see:

```
0D8B:401 20.4A
```

Continued on next page --

ENTER (E)
FILL (F)

Continued from page 11 --

Pressing the ENTER key then ends the Enter function and returns you to the DEBUG prompt sign (-). To confirm all three changes, you may again enter:

ECS:400 followed by three SPACES and see:

0D8B:400 58. 4A. FF. 9E. showing that all three

changes are now in memory.

FILL (F)

Syntax:

Frange list

Purpose:

Fill the addresses in memory in the range with the values in the list.

Additional Notes:

If the range contains more bytes than the number of values in the list, the list will be used repeatedly until all bytes in the range are filled.

If the list contains more values than the number of bytes in the range, the extra values in the list will be ignored.

If any of the memory is bad or nonexistent, they will not be filled and the succeeding positions may be filled with the wrong values, because the Fill function does not abort when an error is encountered as the Enter function does.

If the following command is entered:

F153:200 L100 45 53 41 54 42

DEBUG fills memory locations F153:200 through F153:2FF with the sequence of bytes specified, repeated until all 100H bytes are filled.

Continued on next page --

GO (G)

Syntax:

G[=address1][address2][address3...]

Purpose:

Execute the program currently in memory.

Additional Notes:

If the Go function is entered alone, the program executes as if the program had run outside DEBUG, e.g., in MS-DOS. If optional "=" and address1 is entered, program execution begins at the address specified. If the segment designation is included in address1, both the CS register and the IP register are set to the specified values. If the segment designation is omitted from the command, only the IP register is set. The equal sign (=) enables DEBUG to distinguish the start address (address1) from the breakpoint addresses (address2), (address3), etc.

Up to 10 breakpoint addresses may be specified in one Go command. If you have more than one breakpoint address set, execution will stop at the first breakpoint address in the command that is encountered, regardless of its position in the list of breakpoints, no matter which branch the program takes. When Go is executed, an interrupt code (CCH) is placed at all the specified breakpoint addresses. Then when an instruction at any breakpoint address is encountered, all breakpoint addresses are restored to their original instructions. Thus, Go will stop at only one breakpoint, after which desired breakpoints must be re-entered in another Go command. Breakpoints must be set only at addresses containing the first byte of an 8088 opcode. When program execution reaches a breakpoint, a Register Dump is displayed for the breakpoint address the same as after a Register or Trace command.

The stack pointer must be valid, and there must be at least three words available on the stack for the Go function. The Go function uses an IRET (Return from Interrupt) instruction to cause a jump to the program being executed. The CS, IP and Flag registers are pushed on the stack.

As an example of the Go function, if the following command is entered:

G=7000 7550

Continued on next page --

GO (G)
HEX (H)

Continued from page 13 --

the program executes starting at address 7000H up to the offset address 7550H in the Code Segment, after which the Go function is terminated. If you enter the Go function again, without parameters (after a breakpoint has been encountered) the program execution continues, starting at the breakpoint address, not at the usual starting address, and will continue execution to the normal end-point instruction in the program. If there is no such end-point instruction in the program, DEBUG will go through the end of the program and continue to fetch and execute "garbage" in memory until the system crashes or hangs up.

HEX (H)

Syntax:

Haddress address

Purpose:

Perform hexadecimal arithmetic on the two parameters.

Additional Notes:

DEBUG adds the two parameters (addresses) and then subtracts the second from the first. The sum is displayed, and then the difference, on a single line below the command line. If the difference is a negative number, it will be displayed in two's complement form as a four-digit hex number. For example, if the following command is entered:

H24A 17F the results will be displayed as:

03C9 00CB

If the numbers were reversed, giving a negative difference, you would see:

03C9 FF35

Continued on next page --

INPUT (I)

Syntax:

Ivalue

Purpose:

Input and display one byte from the I/O port specified by value.

Additional Notes:

A 16-bit port address is allowed. As an example of the Input function, if the following command is entered:

I4D6 the byte is input from the port, for example:

72

LOAD (L)

Syntax:

L[address[drive record record]]

Purpose:

Load a file from a disk into memory.

Additional Notes:

The file must have been named, either as a parameter when DEBUG was called up (in the DEBUG command line) or with the Name function after DEBUG is in use. DEBUG always sets BX:CX to the number of bytes loaded, no matter what type file is loaded or where it is loaded into memory. For example, either of the following command sequences will load the (program) file AFILE.COM into memory:

```
A>DEBUG B:AFILE.COM
-L
```

or

Continued on next page --

LOAD (L)

Continued from page 15 --

```
A>DEBUG
          -NB:AFILE.COM
          -L
```

If the Load function is given without parameters, DEBUG loads a .COM file into memory beginning at address CS:100. If the file is an .EXE file, the program is loaded into memory beginning at the load address specified in the header of the .EXE file itself (and the address parameter in the Load command is ignored). Note that the header itself (of the .EXE file) is stripped from the .EXE file before it is loaded into memory. Thus the size of an .EXE file on disk will be larger than its size in memory.

If the Load function is given with an address parameter, loading begins at the memory address specified in the Load command, except for .COM and .EXE files, which are always loaded as stated in the paragraph above.

If Load is entered with the drive and two record parameters (a record is a disk sector, usually 512 bytes), the address parameter must be given, and a number of disk sectors will be loaded from the specified disk drive, given by a hex number corresponding to the drive letter name as follows: 0 = A Drive, 1 = B Drive, 2 = C Drive, etc.

The first record parameter represents the absolute sector number (hex) of the first sector to be loaded on the specified disk. Absolute sector numbers start with 0H and go up to the total number of sectors, less one, on both sides of the disk, which may be many hundreds to several thousand. The second record parameter in the Load command indicates the number (hex) of sectors to be loaded from the disk.

For example: the following command will load 9 sectors (typically 4,608 bytes), numbered AF through B7, from disk Drive B into memory starting at address 1D5F:2000:

```
L1D5F:2000 1 AF 9
```

Note carefully that these parameters do not reference a file, and therefore may cause partial files, or multiple files, or pieces of several files, to be loaded into memory under a single file name, which had to be given to DEBUG prior to using the Load function. For this reason, such data should not be Loaded under a file name already in use.

Continued on next page --

MOVE (M)

Syntax:

Mrange address

Purpose:

Move (copy) the block of memory specified by range to the location beginning at the address specified.

Additional Notes:

Overlapping moves, where part of the new locations overlap some of the current locations are always performed without loss of data. Addresses that could be overwritten are moved first. Moves from higher addresses to lower addresses start at the lowest current address and work up. Moves toward higher addresses start with the highest current addresses and work down.

Note that current data is copied, not moved, and, if not overwritten later, remains the same as it was before the move. Thus, the Move function could also be used to duplicate a block of memory.

As an example of the Move function, if you enter:

MCS:500 630 CS:100

DEBUG first moves address CS:500 to CS:100, then CS:501 to CS:101, and so on until CS:630 is moved to CS:230.

To verify the above move, you could enter DCS:100 and see a memory dump of the data in its new location.

NAME (N)

Syntax:

Nfilespec[filespec...]

Purpose:

Set file names.

Continued on next page --

NAME (N)
OUTPUT (O)

Continued from page 17 --

Additional Notes:

The Name function is used to assign a file name, with an optional drive and/or path name, to a program (file) already loaded into memory by DEBUG or to be loaded by DEBUG. If the file is already loaded, Name will assign a new file name or filespec and cancel the old one. If the file is to be loaded later, the file name or filespec used must exactly match that of the file to be loaded.

Name may be used to assign file name parameters to a file being debugged, for those files which accept or require runtime parameters. Such parameters may include switches that would be valid in a command typed at the MS-DOS system prompt.

For example, if the following sequence of DEBUG commands were entered:

```
-NB:AFILE.EXE
-L
-NA:BFILE.EXE
-G
-W
```

the file AFILE.EXE on the B: Disk would be loaded into memory, its name would be changed to BFILE.EXE, it would be executed, then it would be written on the A: Disk under the new name.

OUTPUT (O)

Syntax:

Ovalue byte

Purpose:

Send the byte specified to the I/O port specified by value.

Additional Notes:

A 16-bit port address is allowed. For example, if you enter:

O3E7 6B

DEBUG outputs the byte 6B to I/O port 3E7.

Continued on next page --

QUIT (Q)

Syntax:

Q

Purpose:

Terminate DEBUG and return to MS-DOS.

Additional Notes:

The Quit function takes no parameters and exits DEBUG without saving the file currently being debugged.

REGISTER (R)

Syntax:

R[register]

Purpose:

Display the contents of one or more 8088/8086 registers. ("Register Dump").

Additional Notes:

If no register parameter is entered, the Register function displays a "Register Dump", i.e., the contents of all microprocessor registers, including all flag registers, followed by the current address in the Code Segment (pointed to by the IP register), the program instruction beginning at that address, in both object code and assembly language mnemonics, and, if the instruction contains a memory reference, the contents of the referenced memory location. For example, this is a Register Dump:

```
R

AX=12FF  BX=0000  CX=FFFF  DX=0000  SP=FFEE  BP=0D5B  SI=0000  DI=1000
DS=0D5B  ES=0D5B  SS=0D5B  CS=0D5B  IP=0100   NV UP EI PL NZ NA PO NC
0D5B:0100 0000          ADD     [BX+SI],AL                      DS:0000=6A
```

Continued on next page --

REGISTER (R)

Continued from page 19 --

If a register name (except the Flag register) is entered, its 16-bit value will be displayed in hex, followed by a colon (:). The colon is a prompt for you to either enter a new value to change the register contents, or simply press the ENTER key if no change is wanted. For example, the following sequence of commands would be used to change the contents of the CX register, and you would see:

```
-RCX
CX FFFF          (current contents of CX = FFFF displayed)
:0000            (new value of 0000 entered by you)
-RCX
CX 0000          (new contents of CX = 0000 verified)
: [ENTER key]  (no further changes desired)
-
```

If you enter:

```
RF
```

DEBUG displays the flags as follows:

```
NV UP DI NG NZ AC PE NC -     (the hyphen is a prompt for input)
```

If you wish to change the contents of one or more of the flag registers , enter the new flag codes (letter pairs), in any order, after the hyphen prompt. For example:

```
NV UP DI NG NZ AC PE NC -ZRPLDN   (three flags to be changed)
-RF
NV DN DI PL ZR AC PE NC -[ENTER key]   (no more changes]
-
```

If you enter two values for the same flag or an invalid flag code, DEBUG will return an error message. In both cases, the flags up to the error will be changed and those after the error will not.

REGISTER FLAG CODES IN DEBUG

FLAG NAME	SET CODE	CLEAR CODE
Overflow	OV Overflow	NV No overflow
Direction	DN Decrement	UP Increment
Interrupt	EI Enabled	DI Disabled
Sign	NG Negative	PL Plus
Zero	ZR Zero	NZ Not zero
Auxiliary	AC Auxiliary carry	NA No aux. carry
Parity	PE Even	PO Odd
Carry	CY Carry	NC No carry

Continued on next page --

SEARCH (S)

Syntax:

Srange list

Purpose:

Search the range specified for the list of bytes (string) specified and print the starting address of the string.

Additional Notes:

The list may contain one or more bytes, each separated by a space or a comma. If the list contains more than one byte, the address of the first byte in the string is returned by DEBUG. If the list contains only one byte, all addresses of the byte in the range are displayed.

For example, when you enter:

SCS:300 3FF AB DEBUG might return:

0D5B:0378
0D5B:03A0
0D5B:03F4

thus indicating three addresses containing the byte ABH.

TRACE (T)

Syntax:

T[=address][value]

Purpose:

Execute one instruction, display a Register Dump showing the contents of all registers resulting from the instruction and the next instruction in the program.

Continued on next page --

TRACE (T)

Continued from page 21 --

Additional Notes:

If the optional address (following the equals sign) is entered, single-stepping ("tracing") begins at the address specified. If the address includes an offset address and a specific value for segment register contents, then both the CS register and the IP register are set to the specified values, which then define where the single-stepping is to begin. If the address refers to the segment base as "CS" (the only segment register allowed) or specifies only an offset address, then the current Code Segment base address (using the current CS register contents) is used to determine the physical address of the instruction. Because the Trace function uses the hardware trace mode of the 8088/8086 microprocessor, instructions from anywhere in memory, including ROM, may be single-stepped through using Trace.

The optional value causes DEBUG to single-step (trace) the number of instructions specified by value, displaying Register Dumps after each instruction is executed. The display scrolls off the top of the CRT screen until the last instruction is executed and the resulting Register Dump is displayed. You may use CTRL-S (hold down CTRL key as you press "S" key) to temporarily halt the scrolling at any point for study, and then restart the display by pressing any key.

The following sequence of commands and displays illustrate the use of the Trace function. If a Register Dump at the current position is as follows:

```
-R

AX=12FF  BX=0000  CX=FFFF  DX=0000  SP=FFEE  BP=0D5B  SI=0000  DI=1000
DS=0D5B  ES=0D5B  SS=0D5B  CS=0D5B  IP=0100   NV UP EI PL NZ NA PO NC
0D5B:0100 2C20          SUB     AL,20
```

```
-T          a Trace command is entered and the instruction SUB  AL,20H
            is executed, giving the following display:

AX=12DF  BX=0000  CX=FFFF  DX=0000  SP=FFEE  BP=0D5B  SI=0000  DI=1000
DS=0D5B  ES=0D5B  SS=0D5B  CS=0D5B  IP=0102   NV UP EI PL NZ NA PO NC
0D5B:0102 8BD8          MOV     BX,AX
```

```
-T=0D9A:0200    this Trace command executes the instruction at
                a totally new physical address (not shown here) and
                displays:

AX=12DF  BX=0000  CX=FFFF  DX=0000  SP=FFEE  BP=0D5B  SI=0000  DI=1000
DS=0D5B  ES=0D5B  SS=0D5B  CS=0D9A  IP=0203   NV UP EI PL NZ NA PO NC
0D9A:0203 F7D0          NOT     AX
```

Continued on next page --

Continued from page 22 --

If you then entered the command T=102 10, DEBUG would execute 16 (10H) instructions, beginning at address 0D9A:0102, displaying a Register Dump after each instruction is executed.

TRACE SPECIAL (P)

Syntax:

P[=address][value]

Purpose:

Same as the Trace function, except for CALL, INT, LOOP and repeating functions (using prefixes REP, REPZ, REPNZ, etc.). For these instructions, Trace Special (P command) executes the entire sequence of individual instructions that these type instructions lead to, treating the entire routine as one instruction. This function displays only one Register Dump, after the last instruction in the routine is executed, no matter how many individual instructions in the routine are executed.

Additional Notes:

The Trace Special (P) function may be used in place of the Trace (T) function for any other type instruction to give the same result as the Trace (T) function.

UNASSEMBLE (U)

Syntax:

U[address][Lvalue]
U[range]

Purpose:

Disassemble object code (machine language) into assembly language source statements and display them both along with addresses of the object code.

Continued on next page --

UNASSEMBLE (U)

Continued from page 23 --

Additional Notes:

The display of disassembled object code looks very similar to an assembly listing produced by an assembler program (e.g., MASM) during the assembly process.

If you enter the Unassemble command without parameters, 20 hex bytes are disassembled to show corresponding assembly language statements, starting at the current address pointed to by the CS and IP registers. If you enter the Unassemble command with the address parameter, then DEBUG disassembles the 20 hex bytes beginning at the address specified. If you enter the Unassemble command with the "L" and value parameter, with or without the address, the number of bytes specified by value will be disassembled rather than the default number of 20.

If you enter the Unassemble command with the range parameter, then DEBUG disassembles all bytes in the address range, whether it is less than or more than 20.

For example, if you enter:

```
05FD:200 208
```

DEBUG will disassemble the bytes between address 05FD:0200 and address 05FD:0208 and you might see:

```
05FD:0200 206472      AND  [SI+72],AH
05FD:0203 69          DB   69
05FD:0204 7665        JBE  016B
05FD:0206 207370      AND  [BP+DI],DH
```

If you enter:

```
UCS:200 L20  or  U200 21F
```

DEBUG will disassemble 32 (20H) bytes starting at offset 0200H in the current Code Segment location, determined by the CS register contents.

If the bytes at some addresses are altered in DEBUG (using the Assemble or Enter function), the Unassemble function can be used to disassemble the revised object code to give revised assembly language statements for use in editing the source program to correspond with the revised object program.

Continued on next page --

WRITE (W)

Syntax:

W[address[drive record record]]

Purpose:

Write the file in DEBUG memory to a disk file.

Additional Notes:

The file must have been named, either as a parameter when DEBUG was called up (in the DEBUG command line) or with the Name function after DEBUG is in use. If the Write command is given without parameters, or with only the address parameter, BX:CX must be set to the number of bytes to be written. This would be the same value as appeared in BX:CX when the file was Loaded, unless the file length has been changed by editing in DEBUG. If you plan to write a file back to disk after revisions in DEBUG, you must keep track of the net number of bytes added or deleted from the file and reset the contents of BX:CX accordingly just before using the Write function. The file in DEBUG is written to the disk from which it was Loaded and overwrites the original file on the disk.

If the Write command is entered with the address parameter, then the file is written beginning at that address. If the address parameter is not used, the file is always written beginning from CS:0100.

If Write is entered with the drive and two record parameters (a record is a disk sector, usually 512 bytes), a number of disk sectors will be written to the specified disk drive given by a hex number corresponding to the drive letter name as follows: 0 = A Drive, 1 = B Drive, 2 = C Drive, etc.

The first record parameter represents the absolute sector number (hex) of the first sector to be written on the specified disk. Absolute sector numbers start with 0H and go up to the total number of sectors, less one, on both sides of the disk, which may be many hundreds to several thousand. The second record parameter in the Write command indicates the number (hex) of sectors to be written to the disk.

For example: the following command will write 9 sectors (typically 4,608 bytes), numbered AF through B7, to disk Drive B, from memory starting at offset address 2000H in the current Code Segment:

```
WCS:2000 1 AF 9
```

Continued on next page --

WRITE (W)
ERROR MESSAGES

Continued from page 25 --

Note carefully that these parameters do not reference any file, and therefore may cause the file being written to overwrite one or more files or pieces of files on the disk and render completely useless the disk directory. For this reason, writing to absolute sectors is **EXTREMELY DANGEROUS** and should be avoided except for very special circumstances!

ERROR MESSAGES

During use of DEBUG, you may receive any of the following error messages. Each error terminates the DEBUG function with which it is associated, but does not terminate the DEBUG program itself or return control to MS-DOS.

ERROR CODE	EXPLANATION
BF	Bad Flag -- The user attempted to alter a flag, but the characters entered were not valid.
BP	Too many Breakpoints -- More than ten breakpoints have been specified as parameters in a Go command. You must re-enter the Go command with ten or fewer breakpoints.
BR	Bad Register -- The Register command was entered with an invalid register name.
DF	Double Flag -- Two values have been entered for one flag register. You may specify a flag value only once per RF command.

END OF INSTRUCTIONS